CITS5501 Software Testing and Quality Assurance
Semester 1, 2018
Workshop 9 – Formal specifications – sample solutions

Model the following systems.

1. An *alarm clock* has two sorts of time it can keep track of: the *current* time, and an *alarm* time.

   It *always* has a current time, and *may* have an alarm time.

One possible answer:

```
sig Time {}

sig AlarmClock {
  currentTime : one Time,
  alarmTime: lone Time
}

pred show( a : AlarmClock ) {
   a.currentTime != a.alarmTime
}

run show for 3 but exactly 1 AlarmClock
```

1. A person can have up to two parents (who are also people). No person is their own ancestor. There is one person – call them bob – who has no parents.

Sample answer:

- Note that the English description can be interpreted a number of ways – does "There is one person ... who has no parents" mean there is *only* one such person, or that there is *at least* one such person? This is exactly the reason we have formal specifications, so we can describe a situation unambiguously.
- Here, we assume "There is one person" means "There is *at least* one person"

```
sig Person {
  parents: set Person
}

fact twoParentsMax {
  all p : Person | #p.parents <= 2
```

```
}

fact noSelfAncestors {
  all p : Person |  not (p in p.^parents)
}

one sig Bob extends Person {
}

fact bobNoParents {
  all b : Bob | no b.parents
}

pred show {
 some p : Person | #p.parents > 1
}

run show for 4
```

1. Entities called *nodes* exist, with which we can make linked lists. A node may have a successor node, called its "*next*" node. Cyclic lists do not exist.

Sample solution:

```
sig Node {
  next: lone Node
}

fact acyclic {
  no n: Node | n in n.^next
}

// we show a case where there are more than 2
// nodes, and at least one head.
//  Alloy will find a situation that satisfies this;
// and we will see that our model doesn't rule out
// a node belonging to multiple lists. (Run the predicate
// to see this.)
pred showOneList(head : Node) {
  no n : Node | n.next = head
  #Node > 2
}

run showOneList for 3
```

1. A *contact list* may contains many *entries*. Each entry must have a "personName" property, and may have telephone, street address, and email properties.

Sample solution:

```
// we'll introduce a string type
// note that String (with an upper case 'S')
// is a reserved word in alloy, so we use lowercase.
sig string {}
```

```
sig Entry {
  personName: string,
  telephone : lone String,
  streetAddress : lone String,
  emaill : lone String
}

sig ContactList {
  entries: set Entry
}


pred show(c: ContactList) {
   #c.entries > 1
   all e : Entry | e in c.entries
}

run show for 4 but exactly 1 ContactList
```